A User Perspective on Sustainable Scientific Software

Brian Blanton and Chris Lenhardt
Renaissance Computing Institute
The University of North Carolina at Chapel Hill

Software is at the core of most modern scientific activities, and as societal awareness of, and impacts from, extreme weather, disasters, and climate and global change continue to increase, scientific software is put more in the spotlight because scientist-developed code is often used to understand, analyze, and predict these types of phenomena. Much of this scientist-coded research software has at least indirect impacts on decision- and policy-making, and so reproducibility of research results becomes an essential component to establishing and maintaining credibility of both scientists and scientific results. This has been highlighted in a recent article by Joppa et al (Troubling Trends in Scientific Software Use, *Science Magazine*, May 2013) that describes reasons for particular software being chosen by scientists, including that the "developer is well-respected" and on "recommendation from a close colleague". This taking of software for granted, assuming that it performs as advertised and that the software itself has been validated and results verified, is one of several big “hazards” facing the entire scientific community.

It is inevitable that scientific software will occasionally be taken for granted. Not all scientists have the same level of expertise in software development, computational sciences, and other related fields. This is hardly a disparagement. As pointed out by Hanney et al (How do Scientists Develop and Use Scientific Software, SECSE '09 *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, 2009), a fundamental difference between science software development and other software development enterprises is that developers of science software generally need some level of knowledge of the science domain. Degrees are granted in the fields of computational sciences and software engineering, and it is unreasonable to expect software engineers to spin up in a specific scientific domain just to write some software. In general, it is much easier for a scientist to acquire the basics of writing working code (whether or not they follow best practices) than for software engineers to become adept in one or more scientific domain. In fact, the latter may not be possible and is probably undesirable. This is partly why so many scientists write their own code.

The issue of taking for granted software essentially becomes, “How should this risk be managed and mitigated?” Hedging against this risk has several aspects, all related to the development of the software itself. In some rough order of increasing complexity, these range from adoption and adherence of software development best practices, to peer-review of software, to formal training in software engineering and related fields for scientists. Ultimately, some balanced approach is needed that encompasses parts of the entire range of approaches, and the goal is to arm scientists

100 Europa Drive Suite 540
Chapel Hill, North Carolina 27517
phone: 919-445-9640
fax: 919-445-9669

with enough best-practices, and provide opportunities for collaborations with professional software developers.

Generally, scientific software begins as a research idea presented to a customer in response to some posited need (as articulated in an RFP, RDFO, RFI, or similar request). The required software is often a by-product of the research efforts. Since the end result of the research is delivery of "science", the software may be written in a get-it-done sense. Some software emerges from this cookery as having substantive value for other applications and research; continued development of the model then occurs in sporadic fits, with multiple developers, and even with students.

Occasionally, a code developed largely in the scientific and engineering research realm gains acceptance and applicability in different sectors like applied research, operations, or commercial and industry use. This does not happen quickly or by coincidence. Years of application of a model develop a history of success, and (presumably) studies conducted with the model demonstrate good predictive skill, good representations of the underlying physics. While not directly peer-reviewed as a software construct, the results have been reasonably well vetted in the community through the publication and peer-review process.

Consider the following issue: Flood insurance is required for federally backed mortgage loans, with rates essentially set according to maps that delineate areas expected to flood with an annual chance of occurrence of 1%. Determining where these zones are requires knowledge of the flood hazard in an area, detailed topographic elevations, and records of observed events for developing the statistical models that represent likely storm occurrences in a region. Most of the statistical and physics-based models used have their origins in research, but the process of conducting the storm surge and statistical study is largely *ad hoc*, even though there are requirements for data management and archiving. There is no requirement on the software design, maintenance, and evolution, even though the ultimate results of a flood insurance study are *regulatory*.

In a coastal flood insurance study, several numerical model codes are used that simulate storm surge and wind-wave responses to tropical cyclones and extra-tropical storms. Because these computational and statistical studies of the coastal environment need to resolve small scale features that place hydraulic constraints on the physical system, we necessarily use models and codes that are themselves research tools into the physics and the numerical/computational methods used to solve the problems on high-performance computers. It is easy to see that this situation (using sophisticated research software for results that become regulatory) contains many of the issues associated with sustainable software development and best-practices adoption and adherence.

So, given the above "use case", the need for sustainable scientific software and development practices extends beyond the credibility argument. Since issues like insufficient documentation, limited test cases, and code unavailability are presumed to be significant barriers to informed and intelligent science software usage, the consensus is that adoption of, and adherence to, best practices in scientific software development will substantially increase intelligent software usage,

100 Europa Drive, Suite 540
Chapel Hill, NC 27517-7583
phone: 919-445-9640
fax: 919-445-9669

thus promoting a sustainable evolution of both the scientific software *and* the science as encoded in the software. Best practices, for example as described by G. Wilson (arXiv:1210.0530v3 [cs.MS]) and others, include designing for people and not computers, defensive programming, optimize only after the code has been validated, use bugs as new test cases, and the use of code versioning.

Perhaps the biggest problem that inhibits development of sustainable science software is the tension and time scale differences that exist between "getting it done" and "getting it done right"? This problem is directly related to how scientific software development is funded, since this directly impacts how the adoption and adherence of best practices can be implemented. Until recently, with the advent of funded software institutes, much scientific software has been developed in an ad hoc manner, with an inconsistent funding stream, and with variable adherence to and application of core software engineering best practices. This situation is exacerbated when the scientist is also the software developer. This could be out of necessity due to resource constraints, or because a scientist *likes* writing code.

Toward one end, scientists become fully engaged in the software development process from inception and design through iterative/agile development and "delivery". This requires substantial planning of research activities into which software best practices and engineering are given equal weight. It is possible that this approach may only be practicable when the science and software engineering are "co-funded". Co-funding, however, implies co-dependence between the groups, which ultimately depends on sustained funding for the *co-development* of scientific software. Sustainability of the software implies and requires sustained funded. Additionally, professional software development is generally expensive and time consuming. Research project budgets generally cannot withstand this level of cost, *unless they are co-funded*. In the academic, scientific area, co-funded development is rare and will likely remain so.

The other "extreme" is for scientists to become relatively expert in software engineering best practices, an approach advocated by some and instanced in several science curricula in the US and Europe. Some level of expertise is essential for scientists to work within the computational community. Many scientists write good code by following best practices, but divergence of the code and intentions are inevitable.

Both of these extremes are relevant and important, primarily because some scientists *like* writing software and would prefer to be intimately involved in its development, and some scientists *don't like* writing code. In terms of sustainable and reusable scientific code, it doesn't matter which path is taken. What matters is that some approach is adopted by individual projects, and the end product is the result of best practices, regardless of who carried out the development.

Fortunately, good software can and does emerge from the relatively ad hoc process of code development, either because at some point a code and supporting infrastructure is completely overhauled with best-practices and software engineering at the forefront, or because some level of good design was adopted early in the process. Many of the numerical models used for coastal ocean research (from process-based studies to forecasting of coastal ocean response to weather

100 Europa Drive, Suite 540
Chapel Hill, NC 27517-7583
phone: 919-445-9640
fax: 919-445-9669

and climate) have been developed in this manner, with differing levels of best-practices strategies (e.g., HYCOM, ROMS, ADCIRC).

One specific idea to increase reproducibility of research results is to adopt a peer-review process for the software used in scientific research, analogous to that in the proposal review and publishing process. If peer-review of scientific results ***and*** the software itself becomes a requirement for publication and is fully implemented, then this would certainly promote early adoption of best practices. There are, however, several factors that complicate this. This implies that a research project and results could be *rejected* if best practices are not adhered to even if the results are sound, This also implies that best-practices *standards* be vetted, approved, and adopted. Who would constitute a set of peers for a review? The pool of potential reviewers that are expert enough in both software engineering and the science domain will be a very small group. How would a review process fit into an agile development cycle? Would each cycle be reviewed? How would this level of involvement be funded? The worry is that only the co-funding model will work, which ultimately does not seem sustainable as a specific project ends but the software continues to evolve and grow.

NSF-funded projects can and should lead the way as to how software can be developed to simultaneously achieve research goals *and* produce sustainable, reusable code. An important step could be for all offices/directorates to recognize and acknowledge the importance of the issue, subsequently require a software development and sustainability plan in the same sense as a data management plan, and (most importantly) enthusiastically fund software (and data management) activities explicitly. And to, of course, adopt standards for both data and software that the software can be valued against.

100 Europa Drive, Suite 540
Chapel Hill, NC 27517-7583
phone: 919-445-9640
fax: 919-445-9669